Coding for the Future: Knights Landing and beyond

CJ Newburn, SW/HW HPC Architect, Community Builder

July 16, 2015 ISC15 IXPUG Workshop:
The Road to Application Performance on Intel® Xeon Phi™ Coprocessor

Outline

A plan for optimization

Trends that are here to stay

New features that involve SW enabling

How development tools help

How analysis tools help

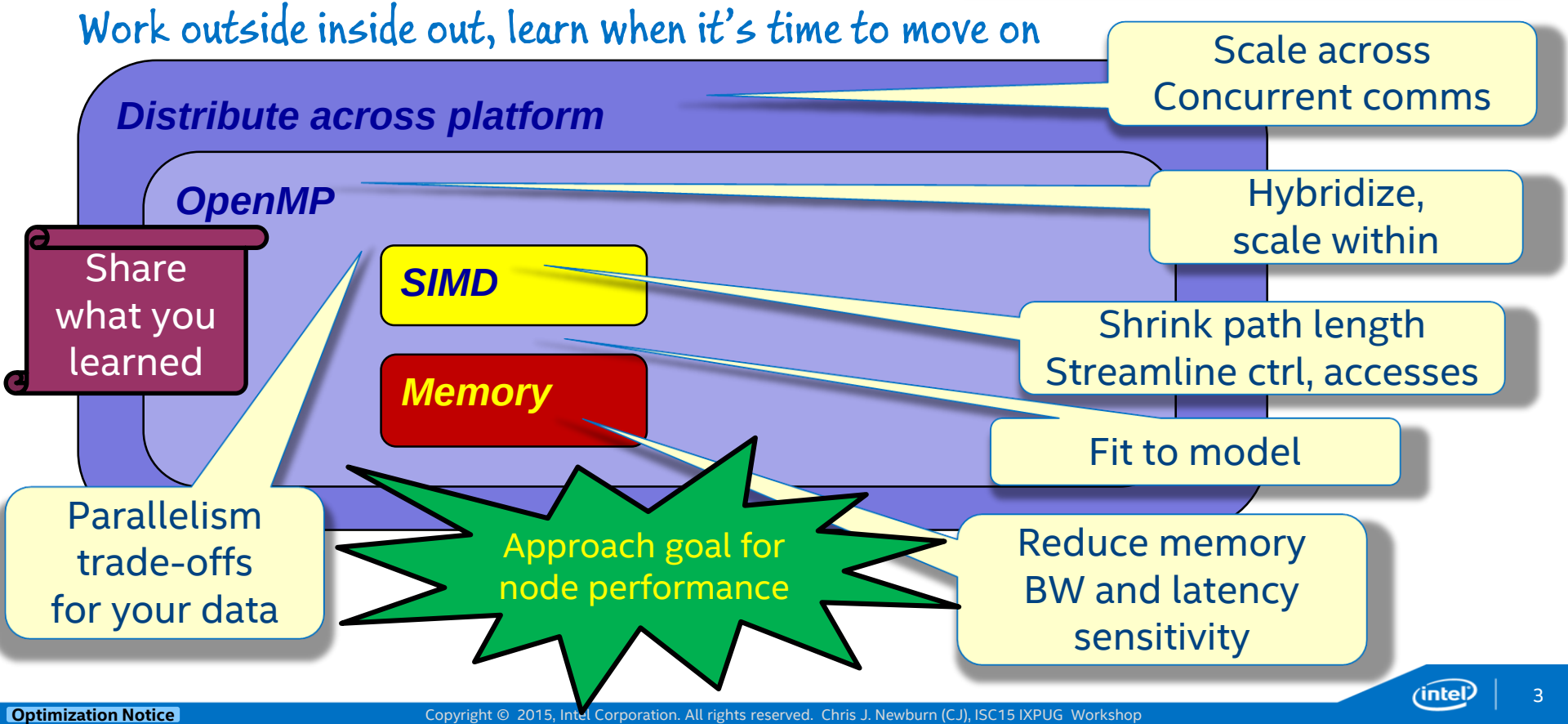
Manual steps

Take-aways

A vision for moving forward

A *plan* for optimization

Work outside inside out, learn when it's time to move on



Trends that are here to stay

Data parallelism

- Lots of threads, spent on MPI ranks or OpenMP/TBB/pthreads
- Improving support for both peak tput and modest/single thread

Bigger, better, faster memory

- High capacity, high bandwidth, low latency DRAM
- Effective caching and paging
- Increasing support for irregular memory refs, modest tuning

ISA innovation

- Increasing support for vectorization, new usages

Fundamental or incremental code changes?

Incremental changes, significant gains ☺

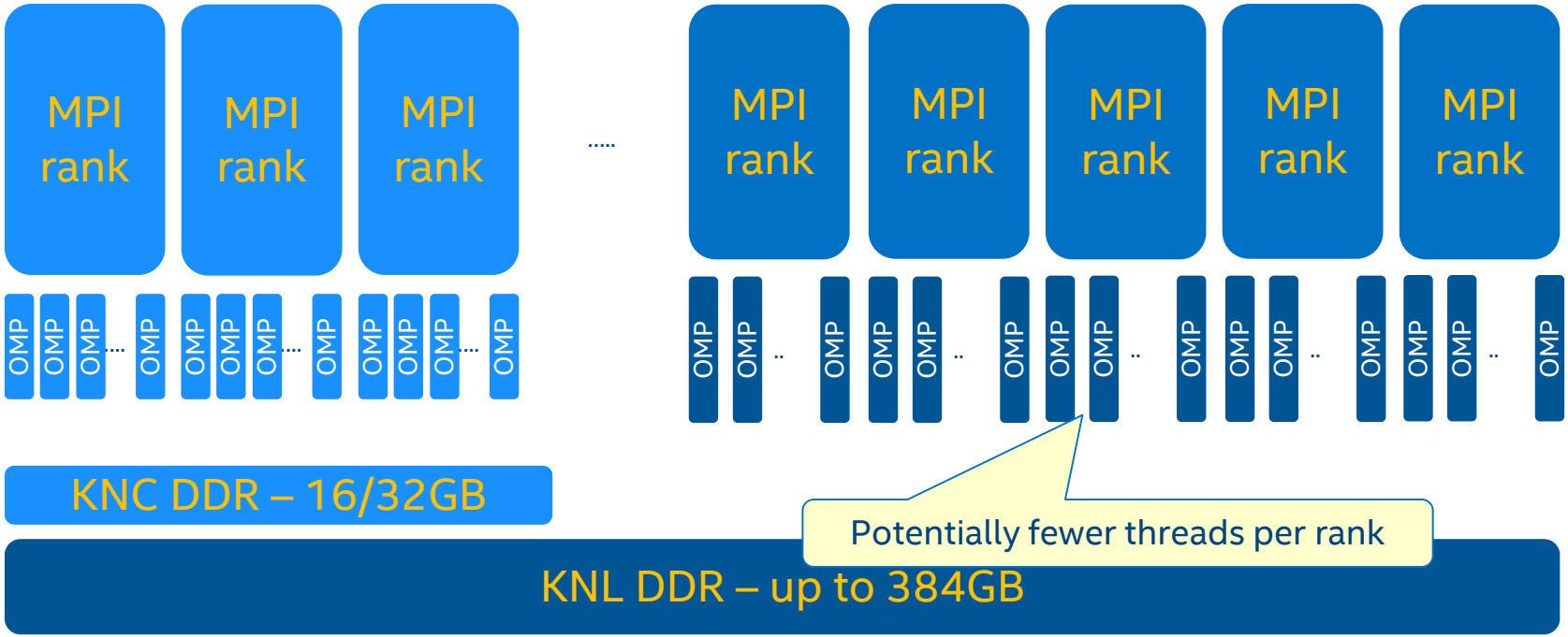
Parallelization – **consistent** strategy

- MPI vs. OpenMP – already needed to tune and tweak
- Less thread-level parallelism required
- Vectorization – **more opportunity, more profitable**

Enable new features with memory tuning

- Access **MCDRAM** with special allocation
- Blocking for MCDRAM vs. just cache

More MPI ranks

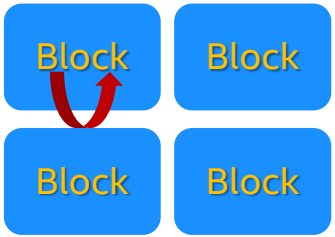


More DRAM enables the use of more MPI ranks

This eases the transition to hybrid MPI+X

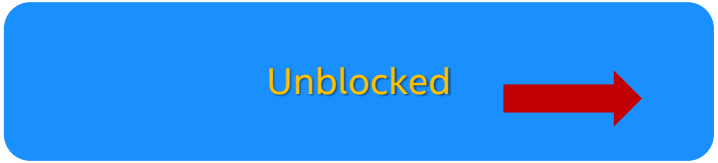
Tolerating a lack of locality, before well tuned

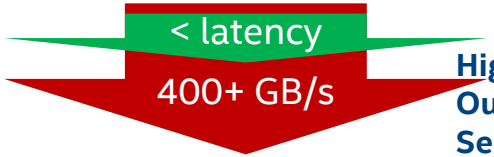
We strongly encourage tuning!



Lower bandwidth
Less sensitive to latency

But until you get there, we help tolerate latency





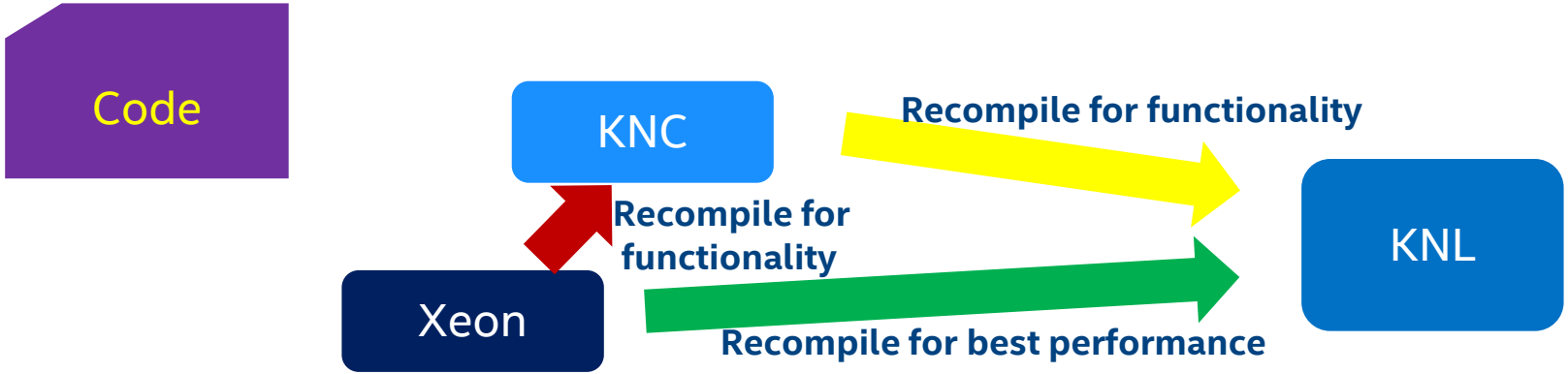
High bandwidth demand
Outstanding misses enqueued
Sensitive to latency

Tuning helps locality: lower BW demand, shorter latency

Until better tuning is achieved, we make the on-ramp gentler:

→ higher BW support, shorter latency to DRAM, more outstanding misses

Compatibility



Binary compatibility with Xeon, recompile for best optimization

More vectorization opportunity: compress/expand, conflicts

Improved vector performance: masking, broadcast

KNL-specific SW enabling

Recompilation, with –xMIC-AVX512

Threading: more MPI ranks, 1 thread/core

Vectorization: increased efficiency

MCDRAM and memory tuning: tile, 1GB pages

Offload continues

How tools help

Development tools

- "Memkind" for MCDRAM allocations
- Intel® Software Development Emulator, v 7.5+
- hStreams supports HBM allocation, task concurrency, offload

Analysis tools

- VTune "memory" analysis for high bandwidth memory analysis

Manual steps

Building

- Change compiler switches in make files

Coding

- Parallelization: vectorization, offload
- Memory management: MCDRAM enumeration and memory allocation

Tuning

- Potentially fewer threads: more cores but less need for SMT
- More memory → more MPI ranks

Take-aways

Keep doing what you were doing for KNC and Xeon

Some goodness comes for free with a recompile

With some extra enabling, use new MCDRAM feature

A vision for how we can move forward

It doesn't take a village… if you want to start from scratch

But if you want us all to spend your time on the fun stuff,

build up the tribal knowledge,

help establish our community's research agenda,

pursue what is awesome

Some examples of thinking bigger

How can you share the good judgment you developed from experience?

Does your work give rise to a collaborative research agenda?

Who could you approach to get traction with?

Filing bugs and feature requests is an investment in the future

- Business-motivated test cases and reproducers are key

Backup

Resources

What's public

MCDRAM

Expand and compress

Conflict detection details

Broadcast details

Overview of tools

References

Porting guide

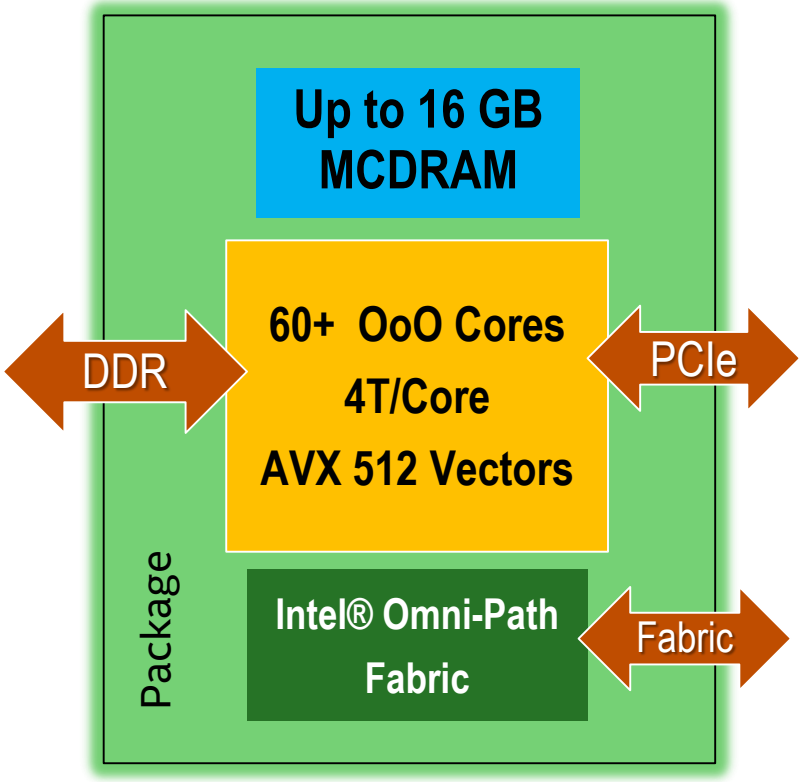
Intrinsics for Intel® AVX-512 instructions : the online document of the User and Reference Guide for the Intel® C++ Compiler

The Intel® Intrinsics Guide : the interactive reference tool for Intel intrinsic instructions

Intel® Architecture Instruction Set Extensions Programming Reference

The Intel Instruction Set Architecture Extensions page: https://software.intel.com/en-us/intel-isa-extensions

Knights Landing Overview



Stand-alone, Self-boot CPU

60+ new Silvermont-based cores

4 Threads per core

AVX 512 vector units

Binary Compatible[1] with Intel® Xeon® processor

2-dimensional Mesh on-die interconnect

MCDRAM: On-Package memory: 400+ GB/s of BW[2]

DDR memory

Intel® Omni-path Fabric

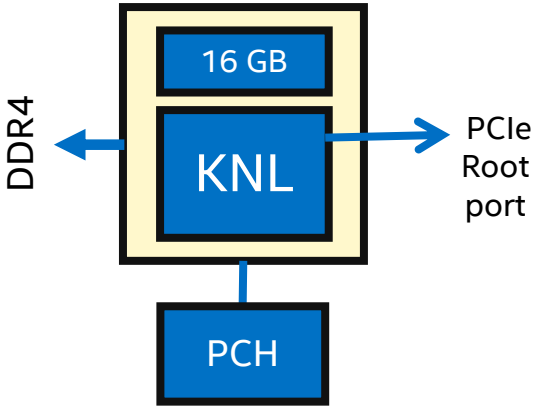
3+ TFLops (DP) peak per package

~3x ST performance over KNC

Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). 2Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as flat memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

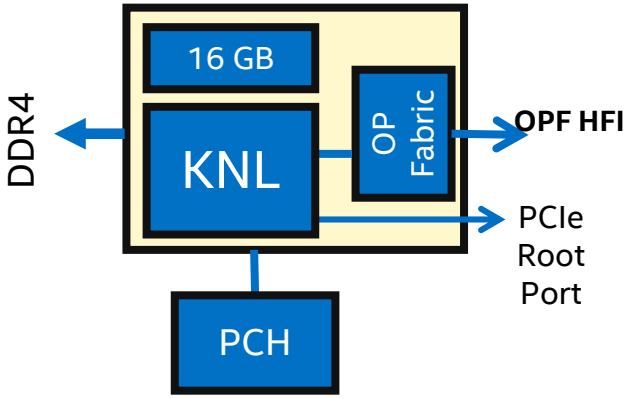
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance

Knights Landing Products



KNL

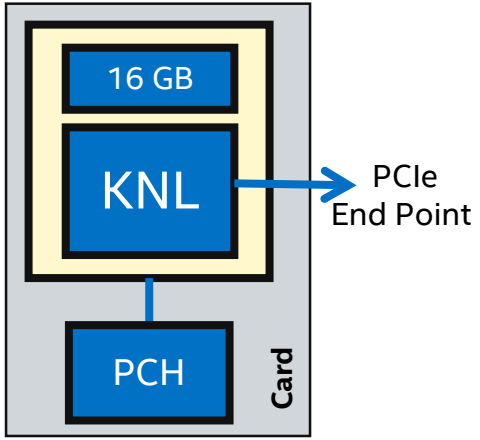
DDR
MCDRAM: up to 16 GB
Gen3 PCIe (Root port)



KNL with Fabric

DDR
MCDRAM: up to 16 GB
Gen3 PCIe (Root port)
Omni-Path Fabric

Self Boot Socket



KNL Card

No DDR Channels
MCDRAM: up to 16 GB
Gen3 PCIe (End point)

PCIe Card

Public info, 1

- **Highlights**
  - Massive thread parallelism, data parallelism, mem bw
  - Impr single-thread, Xeon binary compatibility (Hsw-TSX)
  - Intel® Omni-Path™ fabric integration (KNL-F w/ Linux)
  - CPU and traditional PCIe coprocessor form factor
- **Microarchitecture**
  - Over 8 billion transistors per die based on Intel's 14 nanometer manufacturing technology
  - Binary compatible with Intel® Xeon® Processors with support for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) (Hsw – TSX)
  - 3x Single-Thread Performance compared to 1st generation Intel® Xeon Phi™ coprocessors (7120P)
  - 60+ cores in a 2D Mesh architecture
  - 2 cores per tile with 2 vector processing units per core
  - 1MB L2 cache shared between 2 cores in a tile (cache-coherent)
  - Most of today's parallel optimizations carry forward to Knights Landing
  - Multiple NUMA domain support per socket
- **Based on Silvermont**
  - 4 Threads / Core
  - 2X Out-of-Order Buffer Depth (vs. Silvermont)
  - Gather/scatter in hardware, advanced Branch Prediction
  - High cache bandwidth, 32KB Icache, Dcache
  - 2 x 64B load ports in Dcache
  - 46/48 Physical/Virtual Address bits
- **MCDRAM**
  - Over 5x STREAM vs. DDR4[1] ⇒ Over 400 GB/s
  - Up to 16GB at launch
  - NUMA support
  - Over 5x Energy Efficiency vs. GDDR5[2]
  - Over 3x Density vs. GDDR5[2]
  - In partnership with Micron Technology
  - Flexible memory modes including cache and flat
- **Server**
  - DDR4 using 6 channels
  - Reliability ("Intel server-class reliability")
  - Power Efficiency (Over 25% better than discrete coprocessor) ⇒ Over 10 GF/W
  - Density (3+ KNL with fabric in 1U)5
  - Up to 36 lanes PCIe* Gen 3.0
- **Availability**
  - First commercial HPC systems in 2H'15
  - 1st generation Intel® Xeon Phi™ coprocessor to Knights Landing upgrade program available today
  - Intel Adams Pass board (1U half-width) is custom designed for Knights Landing (KNL) and will be available to system integrators for KNL launch; the board is OCP Open Rack 1.0 compliant, features 6 ch native DDR4 (1866/2133/2400MHz) and 36 lanes of integrated PCIe* Gen 3 I/O

https://software.intel.com/en-us/articles/what-disclosures-has-intel-made-about-knights-landing

Public info, 2

- **Availability**
  - First commercial HPC systems in 2H'15
  - 1st generation Intel® Xeon Phi™ coprocessor to Knights Landing upgrade program available today
  - Intel Adams Pass board (1U half-width) is custom designed for Knights Landing (KNL) and will be available to system integrators for KNL launch; the board is OCP Open Rack 1.0 compliant, features 6 ch native DDR4 (1866/2133/2400MHz) and 36 lanes of integrated PCIe* Gen 3 I/O
- **Momentum**
  - Cori Supercomputer at NERSC (National Energy Research Scientific Computing Center at LBNL/DOE) became the first publically announced Knights Landing based system, with over 9,300 nodes slated to be deployed in mid-2016
  - "Trinity" Supercomputer at NNSA (National Nuclear Security Administration) is a $174 million deal awarded to Cray that will feature Haswell and Knights Landing, with acceptance phases in both late-2015 and 2016.
  - Expecting over 50 system providers for the KNL host processor, in addition to many more PCIe*-card based solutions.
  - >100 Petaflops of committed customer deals to date
  - The DOE* and Argonne* awarded Intel contracts for two systems (Theta and Aurora) as a part of the CORAL* program, with a combined value of over $200 million. Intel is teaming with Cray* on both systems. Scheduled for 2016, Theta is the first system with greater than 8.5 petaFLOP/s and more than 2,500 nodes, featuring the Intel® Xeon Phi™ processor (Knights Landing), Cray* Aries* interconnect and Cray's* XC* supercomputing platform. Scheduled for 2018, Aurora is the second and largest system with 180-450 petaFLOP/s and approximately 50,000 nodes, featuring the next-generation Intel® Xeon Phi™ processor (Knights Hill), 2nd generation Intel® Omni-Path fabric, Cray's* Shasta* platform, and a new memory hierarchy composed of Intel Lustre, Burst Buffer Storage, and persistent memory through high bandwidth on-package memory.
- **Future**
  - Knights Hill is the codename for the 3rd generation of the Intel® Xeon Phi™ product family
  - Based on Intel's 10 nanometer manufacturing technology
  - Integrated 2nd generation Intel® Omni-Path Host Fabric Interface

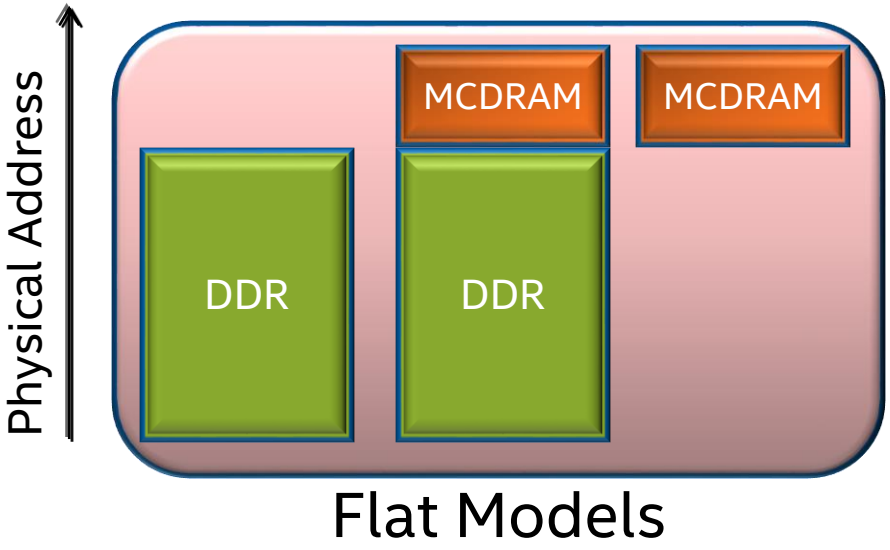
Beyond AVX-512 Foundation

- Intel AVX-512 Prefetch Instructions (PFI)
- Intel AVX-512 Exponential and Reciprocal Instructions (ERI)
- Intel AVX-512 Conflict Detection Instructions (CDI)

CPUID	Instructions	Description
AVX512PF	PREFETCHWT1	Prefetch cache line into the L2 cache with intent to write
	VGATHERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache
	VSCATTERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache with intent to write
AVX512ER	VEXP2{PS,PD}	Computes approximation of 2^x with maximum relative error of 2^{-23}
	VRCP28{PS,PD}	Computes approximation of reciprocal with max relative error of 2^{-28} before rounding
	VRSQRT28{PS,PD}	Computes approximation of reciprocal square root with max relative error of 2^{-28} before rounding
AVX512CD	VPCONFLICT{D,Q}	Detect duplicate values within a vector and create conflict-free subsets
	VPLZCNT{D,Q}	Count the number of leading zero bits in each element
	VPBROADCASTM{B2Q,W2D}	Broadcast vector mask into vector elements

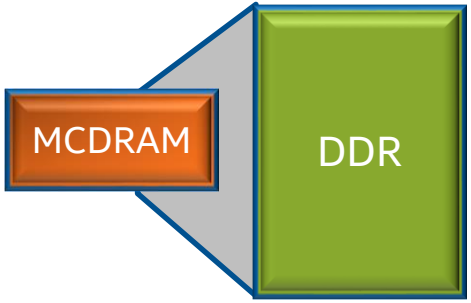
3 Memory Modes

- Mode selected at boot
- MCDRAM-Cache covers all DDR

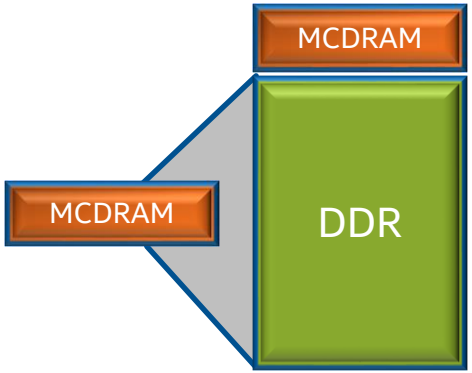


Flat Models

Cache Model

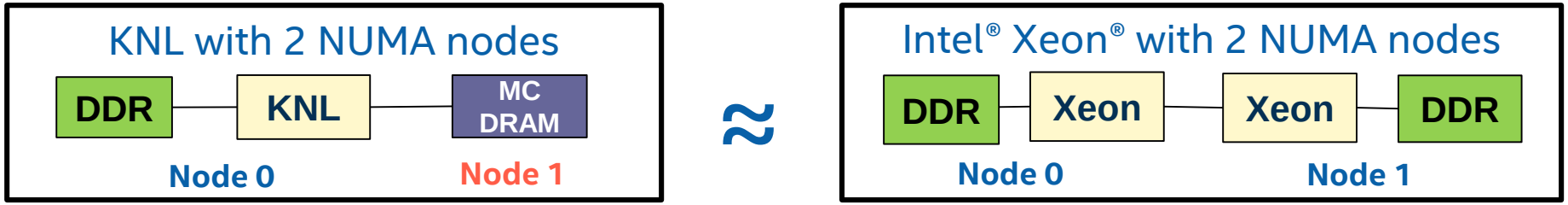


Hybrid Model



Flat MCDRAM: SW Architecture

MCDRAM exposed as a separate NUMA node



Memory allocated in DDR by default

- Keeps low bandwidth data out of MCDRAM.

Apps explicitly allocate important data in MCDRAM

- **"Fast Malloc"** functions: Built using NUMA allocations functions
- **"Fast Memory"** Compiler Annotation: For use in Fortran.

Flat MCDRAM using existing NUMA support in Legacy OS

Flat MCDRAM SW Usage: Code Snippets

Allocate 1000 floats from DDR

```
float   *fv;

fv = (float *)malloc(sizeof(float) * 1000);
```

Allocate 1000 floats from MCDRAM

```
float   *fv;

fv = (float *)hbw_malloc(sizeof(float) * 1000);
```

Allocate arrays from MCDRAM & DDR in Intel FORTRAN

```
c      Declare arrays to be dynamic
       REAL, ALLOCATABLE :: A(:), B(:), C(:)

!DEC$ ATTRIBUTES, FASTMEM :: A

       NSIZE=1024
c
c      allocate array 'A' from MCDRAM
c
       ALLOCATE (A(1:NSIZE))
c
c      Allocate arrays that will come from DDR
c
       ALLOCATE  (B(NSIZE), C(NSIZE))
```

Keeping the App Effort Level Low

High Bandwidth (HBW) Malloc API

```
HBWMALLOC(3)                HBWMALLOC                HBWMALLOC(3)

NAME
       hbwmalloc - The high bandwidth memory interface

SYNOPSIS
       #include <hbwmalloc.h>

       Link with -ljemalloc -lnuma -lmemkind -lpthread

       int hbw_check_available(void);
       void* hbw_malloc(size_t size);
       void* hbw_calloc(size_t nmemb, size_t size);
       void* hbw_realloc (void *ptr, size_t size);
       void hbw_free(void *ptr);
       int hbw_posix_memalign(void **memptr, size_t alignment, size_t size);
       int hbw_posix_memalign_psize(void **memptr, size_t alignment, size_t size, int pagesize);
       int hbw_get_policy(void);
       void hbw_set_policy(int mode);
```

Publicly released at https://github.com/memkind

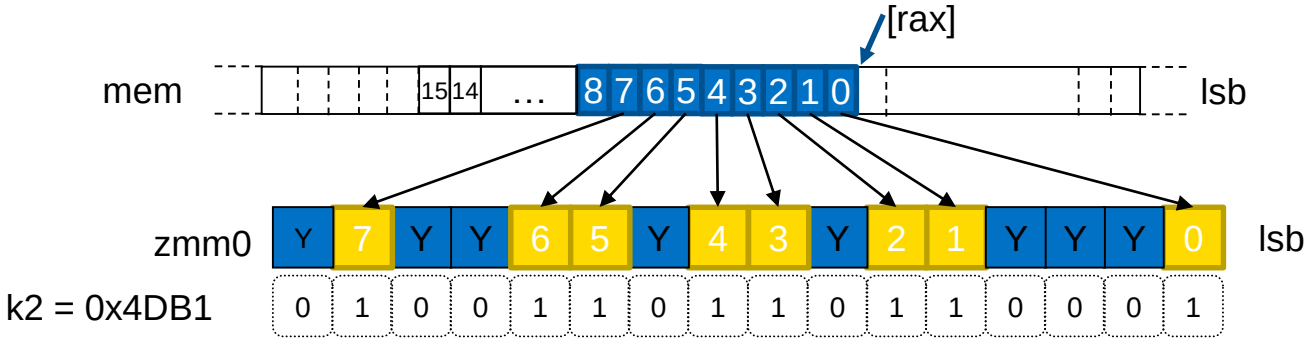
Expand & Compress

Allows vectorization of conditional loops
- Opposite operation (compress) in AVX512F
- Similar to FORTRAN pack/unpack intrinsics
- Provides memory fault suppression
- Faster than alternative gather/scatter

```
for(j=0, i=0; i<N; i++) {
  if(C[i] != 0.0)  {
     B[i] = A[i] * C[j++];
  }
}
```

VEXPANDPS zmm0 {k2}, [rax]

Moves compressed (consecutive) elements in register or memory to sparse elements in register (controlled by mask), with merging or zeroing



Motivation for Conflict Detection

Sparse computations are common in HPC, but hard to vectorize due to race conditions

Consider the “histogram” problem:

```
for(i=0; i<16; i++) {  A[B[i]]++; }
```

- Code above is wrong if any values within B[i] are duplicated
  - Only one update from the repeated index would be registered!
- A solution to the problem would be to avoid executing the sequence gather-op-scatter with vector of indexes that contain conflicts

Conflict Detection Instructions in AVX-512

VPCONFLICT instruction detects elements with previous conflicts in a vector of indexes

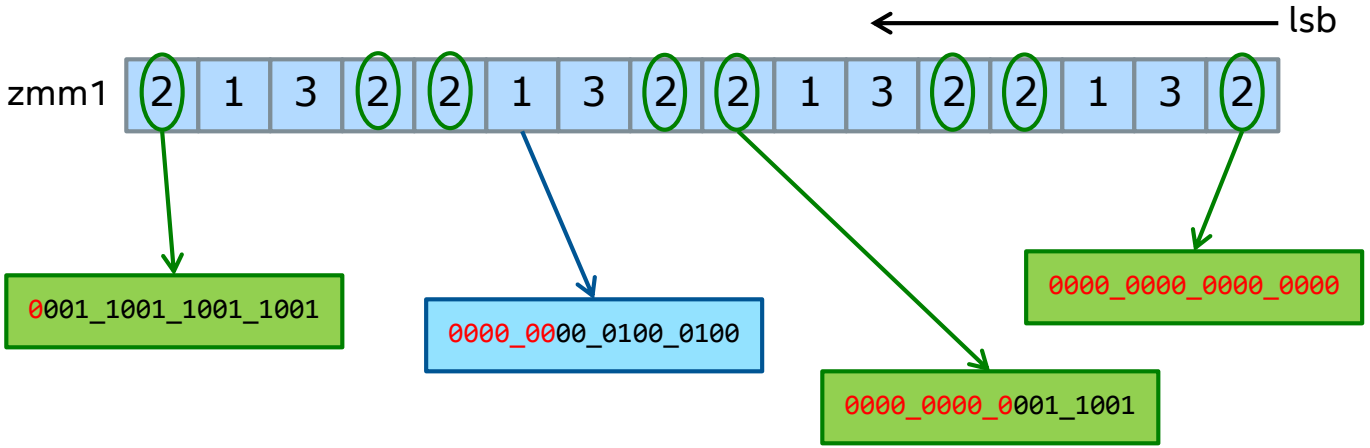
- Allows to generate a mask with a subset of elements that are guaranteed to be conflict free
- The computation loop can be re-executed with the remaining elements until all the indexes have been operated upon

CDI instr.
VPCONFLICT{D,Q} zmm1{k1}, zmm2/mem
VPBROADCASTM{W2D,B2Q} zmm1, k2
VPTESTNM{D,Q} k2{k1}, zmm2, zmm3/mem
VPLZCNT{D,Q} zmm1 {k1}, zmm2/mem

```
index = vload &B[i]                                  // Load 16 B[i]
pending_elem = 0xFFFF;                               // all still
remaining
do {
   curr_elem = get_conflict_free_subset(index, pending_elem)
   old_val = vgather {curr_elem} A, index            // Grab A[B[i]]
   new_val = vadd old_val, +1.0                      // Compute new
values
   vscatter A {curr_elem}, index, new_val            // Update A[B[i]]
   pending_elem = pending_elem ^ curr_elem           // remove done idx
} while (pending_elem)
```

VPCONFLICT{D,Q}

- VPCONFLICT{D,Q} zmm1{k1}{z}, zmm2/B(mV)
  - For every element in ZMM2, compare it against everybody and generate a mask identifying the matches (but ignoring elements to the 'left' of the current one –i.e. "newer")
    - Store every mask in every element destination in ZMM1



Optimized Algorithm

```
for each 16 scalar iterations {
   indices = vload &index_array[i]
   vpconflictd comparisons, indices
   vplzcntd tmp_lzcnt, comparisons
   vpsubd perm_idx, all_31s, tmp_lzcnt
   temp_values = do_first_iteration(); // gather + compute
   vptestmd to_do {k0}, comparisons, all_ones // anything left?
   while (to_do) {
      vpbroadcastmd tmp, to_do
      vptestnmd mask {to_do}, comparisons, tmp
      vpermd tmp_values {mask}, perm_idx
      tmp_values = do_work(mask); // just compute!
      to_do ^= mask;
   } while(to_do);
   vscatter indices, A, tmp_values
}
```

Obtain recurrence indices

Store results

Re-do conflicting indices reusing results directly from the vector

Embedded Broadcasts, Masking Support

VFMADD231PS zmm1, zmm2, C {1to16}

- Scalars *from memory* are first class citizens
  - Broadcast one scalar from memory into all vector elements before operation
- Memory fault suppression avoids fetching the scalar if no mask bit is set to 1

```
float32 A[N], B[N], C;

for(i=0; i<8; i++)
{
    if(A[i]!=0.0)
      A[i] = A[i] + C* B[i];
}
```

Other “tuples” supported

- Memory only touched if at least one consumer lane needs the data
- For instance, when broadcast a tuple of 4 elements, the semantics check for every element being really used
  - E.g.: element 1 checks for mask bits 1, 5, 9, 13, …

```
VBROADCASTSS zmm1 {k1}, [rax]
VBROADCASTF64X2 zmm2 {k1}, [rax]
VBROADCASTF32X4 zmm3 {k1}, [rax]
VBROADCASTF32X8 zmm4, {k1}, [rax]
…
```

Tools overview

MPI
- development: Intel MPI
- quick summary: MPS
- drill down, correctness: ITAC

Threads
- development: OpenMP, TBB, Cilk
- correctness: Inspector
- parallelism: Advisor
- scaling: VTune

Memory
- analysis: MPS, VTune, Advisor

Libraries
- math: MKL
- data analytics: DAAL
- media: IPP

Vectors
- development: C++/Fortran Compilers
- parallelism: Advisor
- analysis: VTune, Advisor
- diagnostics: compiler reports, Advisor

Legal Disclaimer & Optimization Notice

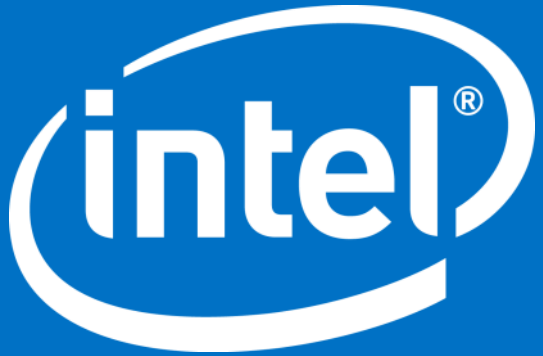
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel®